Itrodução a FORTRAN

Rodrigues, J. L.

2020

Sumário

1 Elementos Básicos em FORTRAN

Nesta seção o aluno deverá aprender os seguintes temas:

- Quais caracteres são permitidos em uma sentença
- Conhecer a estrutura básica de uma declaração e programa em **FORTRAN**
- Reconhecer a diferença entre declarações executáveis e não executáveis
- Entender a diferença entre constantes e variáveis
- Saber diferenciar *INTEGER, REAL, CHARACTER*, ou seja, os tipos de dados
- Aprender a diferença entre *types(tipos): padrão(Default) e explícita(explicit)* e ter a compreensão de porquê *explicit* deve sempre ser usado
- Conhecer a estrutura de uma declaração de atribuição
- Conhecer a hierarquia ou a sequência das operações
- Entender como FORTRAN trabalha com *modo-misto* de operações aritméticas
- Entender como são as funções intrínsecas, e como usá-las
- O por quê de usar a declaração ***IMPLICIT NONE***

Ao final dessa seção, é interessante que o leitor volte nessa lista e confira seus conhecimentos!

1.1 Introdução

É comum na física, principalmente na áreas de simulação, usar a linguagem FORTRAN por ser excelente para lhe dar com cálculos de números de forma rápida e precisa. Aqui será introduzido as bases da linguagem e como em qualquer área, as bases são imprescindíveis!

1.2 Signos de FORTRAN

Toda linguagem é formada por signos, com uma linguagem de computação não seria diferente. Em FORTRAN, seus signos são conhecidos como ***FORTRAN character set***. Consiste em 97 caracteres mostrados na tabela 1:

Tabela 1: ***FORTRAN character set***

26 maiúsculas	A-Z
26 minúsculas	a-z
10 números	0-9
1 underscore	”_”
5 símbolos de aritmética	+ - * / **
28 símbolos diversos	( ) . = , ’ $: ! ”% & ; ¡ ¿ ? [ ] ‘ ^ — # @ *e espaço(barra de espaço).*

FORTRAN é uma linguagem *case insensitive* o que é o mesmo que dizer que ela não faz distinção entre letras maiúsculas e minúsculas.

1.3 Estrutura de uma declaração em FORTRAN

As declarações em FORTRAN seguem dois tipos básicos:

- Executáveis
- Não executáveis

* Os executáveis são ações que o programa realizará, são, operações aritméticas, leitura de dados, etc...

* Os não executáveis são os que nos dão informação sobre o programa ou do que está acontecendo naquela parte do código.

As declarações podem ser feitas em qualquer lugar, no geral, FORTRAN em comparação com as demais linguagens é *"mais livre"*, porém tem um limite máximo de 132 caracteres por linha, caso você exceda esse limite pode-se usar simbolo (&) para continuar na próxima linha. Um exemplo está na tabela 2 onde as três declarações são idênticas:

Tabela 2:

x = a + b
x = a & + b
9x = a & & + b

As declarações acima, somam a e b que podem ser variáveis com valor pré-definido ou a qual o usuário do código fornecerá que será salva na variável x. No geral, você pode continuar o seu código até 256 caracteres.

Você deve ter notado que na última declaração a variável começa com o número 9, esse número é chamado de *statement label* ou em português *rótulo da declaração*, ele é praticamente um nome que se dá a declaração e deve ser um número que esteja entre 1 e 99999. O statement label não possui nenhuma informação sobre a declaração além de seu nome, e pode ser usado apenas 1 vez durante o código, é único. É bem raro encontrá-lo nos códigos atuais, e se for usado deve ser usado uma vez e apenas uma vez na mesma unidade do programa.

Em FORTRAN, o simbolo "!"é usado como um símbolo de comentário sendo que o compilador o despreza quando estiver gerando o programa executável. Ou seja, tudo que começar com "!"é um comentário que está no código. *Comentários são extremamente importantes, pois ajudam ao programador como também seus pares a entender o que está sendo feito ali, portanto todo código deve ser comentado.*

```
!este é um exemplo de comentário em FORTRAN
```

1.4 Estrutura básica de um programa em FORTRAN

Um programa simples em fortran é:

Tabela 3: exemplo primeiro programa

```
program primeiro_programa
!objetivo:
!mostrar um programa basico em FORTRAN
!declaração das variaveis que serao usadas nesse programa
!todas as variaveis sao inteiras
integer:: x,y,z
write(*,*)'entre com duas variaveis inteiras'
read(*,*)x,y
!multiplicacao das variav
z = x*y
!escrever o resultado
write(*,*)'resultado = ', z
!para o primeiro programa n poderia dxr de fora
write(*,*)'Ola mundo, hello world!'
stop
end program primeiro programa
```

O exemplo descrito na tabela 3 é bom para primeiro programa, pois, está dividido em três seções:

1. Declaração não-executável: Passos não executáveis, como o nome do programa e seus comentários
2. Declaração executável: Uma ou mais declarações descrevendo as ações que serão performadas pelo programa.
3. Chamada de finalização: serve para dizer ao compilador para parar e que o programa chegou ao fim.

1.4.1 Declaração não-executável

A declaração como dito anteriormente, são, comandos que não executam nenhuma ação. Nomeia o programa como também define as variáveis do mesmo. A primeira declaração é *program*, um não-executável o qual define o nome do programa para o compilador. O nome pode ter até 63 caracteres, alfanuméricos, underscores(_), et cetera. É imprescindível que o primeiro caractere seja intrinsecamente uma letra a-z, A-Z. Após, temos os comentários que descrevem o propósito do programa. Depois, vem a declaração *integer*, é um não-executável que será destrinchado melhor daqui a pouco, mas por hora, nesse código ela serve para declarar as variáveis inteiras: $x, y, z \in \mathbb{Z}$.

1.4.2 Declaração executável

É onde uma ou mais ações programadas serão executadas pelo compilador. O primeiro executável nesse código é a declaração *write*, que, neste caso escreve no terminal ou prompt a mensagem, número, et cetera. O próximo é a declaração *read* que vai ler os números que o usuário escrever. A próxima é a operação z = x*y que irá multiplicar as variáveis x e y uma pela outra. Por fim temos o *write* novamente.

1.4.3 Chamada de finalização

Consiste nos comandos *stop, end program*. A primeira diz quando parar o programa e a segunda anuncia onde termina o programa.

Usando apenas *stop* o programa irá parar a execução. Usando *stop* 3, ou seja, com um número, o programa irá imprimir na tela o número que também será retornado ao sistema como um código de erro. Se, *stop 'Erro de parada'*, é usado com uma *string*, isto é, com um corpo textual, a *string* será printada na tela quando o programa parar. Contudo, o uso do stop é **opcional!!!** O programa gera um stop automático ao ler 'end program' seu uso é raro, mas tem de ser mencionado.

Há um stop adcional chamado de *error stop*, o qual foi adicionado no FORTRAN 2008, em que ele para o programa se houver algum erro de execução.

1.4.4 Estilo de programa

Há estilos de programas em que usurários fazem o uso letras maiúsculas em palavras chaves *PROGRAM, WRITE, READ, INTEGER* e usam minúsculas para as variáveis, como também maiúsculas para constantes PI(π). Isso não é um requisito! Haja vista que FORTRAN é *case insentive* como antes mencionado.

Dica!
Adote para si um estilo de programação e use-o em todos os programas
Se familiarize com seu estilo!!!

1.4.5 Compilando, linking e executando o programa em FORTRAN

Depois de escrever o código, ele precisa ser compilado e depois *linkado(Link em inglês significa nesse caso ligar, linkado é o mesmo de dizer ligado, porém usado especificamente no âmbito computacional)* a bibliotecas no computador para produzir um programa executável.

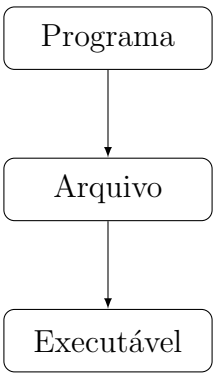


Figura 1: Fluxograma passos de código

No fluxograma acima, a compilação é o passo entre o programa e o arquivo e o link é o passo entre o arquivo e o executável.

Os programas em FORTRAN podem ser compilados, linkados e executados em dois modos **batch** e **interactive**. No modo batch, ou seja, em lotes(tradução de batch) o

programa é executado sem uma entrada ou interação com o usuário, era bastante utilizado no passado.

O programa no modo interativo(interactive), funciona de modo a pedir uma entrada do usuário. Hoje em dia é o mais utilizado, porém o número de programas em batch é muito grande sendo de interesse aprender as duas formas.

1.5 Constantes e variáveis

Uma **constante** é um dado definido antes da execução do programa e durante o processo não troca o seu valor. Quando o compilador encontra a constante ele guarda em um local na memória e referencia aquele local sempre que a constante é chamada.

Uma **variável** é um tipo de dado que pode sempre trocar o seu valor durante a execução do programa, quando o compilador encontra uma variável ele a guarda em um local na memória e referencia aquele local sempre que a variável é chamada.

Cada variável deve ter um nome único, pois o programa irá guardar essa variável em um local específico na memória. As variáveis podem ser nomeadas com no máximo 63 caracteres, podem conter alfanuméricos e underscores(_), porém, devem sempre começar com letras a-z, A-Z.

É importante colocar nomes nas variáveis que tenham significado, por exemplo se o programa for calcular uma data é bom que as variáveis sejam, *dia, mes, ano*.

Dica!
sempre que puder use nome significativos no seu código!!!

Se faz também de enorme importância um dicionário dos dados do programa no cabeçalho do seu código. É importante que liste a definição de cada variável usada no programa, descrição e conteúdo de cada item bem como as unidades em que eles estão. Pode parecer desnecessário durante a programação, mas para um código grande, quando há de se fazer uma modificação, é vital esse dicionário!

Dica!
Use sempre um dicionário de dados no cabeçalho do programa!!!

Em FORTRAN, há cinco tipos intrínsecos de 'dados construtores' que são constantes ou variáveis. Três numéricos(***INTEGER, REAL, COMPLEX***), um lógico(***LOGICAL***) e um para strings(***CHARACTER***).

Os tipos de dados INTEGER, REAL e CHARACTER serão discutidos por agora enquanto que o restante somente nos próximos capítulos.

1.5.1 INTEGER constantes e variáveis

O dado INTEGER é constituído de constantes e variáveis e só pode guardar valores inteiros $n \in \mathbb{Z}$. Uma constante INTEGER é um número que não possui ponto decimal. Se é positivo pode ser escrito com ou sem o sinal +, se for negativo tem que colocar '-' na frente do número.

Uma variável INTEGER só pode conter valores inteiros.

Geralmente as constantes e variáveis INTEGER são armazenadas em somente uma palavra no computador, como a arquitetura varia de 32 para 64 bits nos computadores, o maior inteiro também varia. Quase todos os compiladores de FORTRAN tem suporte para diferentes tamanhos de inteiros, são conhecidos como diferentes "tipos"de inteiros usados no código como **kind**, que será abordado mais adiante.

1.5.2 REAL constantes e variáveis

O tipo de dado REAL consiste em um armazenamento nos formatos reais, $\mathbb{R}$, ou pontos flutuantes. Diferentemente dos INTEGER's, REAL pode representar números com componentes fracionais.

Uma constante REAL tem que ser escrita com um ponto decimal, acompanhado ou não de um expoente, se a constante é positiva pode ou não se colocar +, é opcional, e se for negativa é obrigatório o sinal -". O tipo de dado REAL não trabalha com vírgulas. Se o uso de um expoente se fizer necessário pode ser usado a letra E seguida por um expoente positivo ou negativo. A *mantissa*(a parte antes do expoente) deve conter ponto decimal exemplos:

10.

−999.9

+1.0E-3

123.45E20

0.12E+1

Uma variável REAL é uma variável que contém um valor real $x \in \mathbb{R}$. O seu armazenamento funciona divido em duas partes: a mantissa e o expoente. O número de bits que alocará a mantissa determinará a *precisão* da constante enquanto o número de bits alocado ao expoente determina o *range* da constante.

- Precisão: Número de dígitos significativos na qual a constante é conhecida
- Range: A faixa de maior e menor valor a qual pode ser representado.

A tabela abaixo representa valores para um computador padrão **IEEE 754**

Tabela 4: Precisão e Range dos números reais

Nº total bits	Nº bits mantissa	Precisão decimais	Nº bits no expoente	Range expoente
32	24	7	8	10^{-38} a 10^{38}
64	53	15	11	10^{-308} a 10^{308}
128	112	34	16	10^{-4932} a 10^{4932}

No geral, os compiladores de FORTRAN trabalham tanto em 32 quanto 64 bits, são chamados de **kind**. Se você escolher o kind apropriado, é possível aumentar a precisão e o range da constante ou variável REAL. A linguagem é munida de um mecanismo explícito para tal coisa.

1.5.3 constante e variáveis CHARACTER

O tipo de dado CHARACTER consiste em uma string de caracteres alfanuméricos. Uma constante é encapsulada dentro de (") ou ('), simples ou duplo. O mínimo de caracteres que pode ser usado é zero, e o máximo varia de compilador para compilador.

Os caracteres dentro de (') ou (") estão dentro do contexto de caracteres, ou seja, todos os caracteres representados em um computador é um caractere válido, não somente os 97 apresentados na tabela 1. Caso haja a necessidade de incluir uma apóstrofe na string ela pode ser representada como duas apóstrofes em sequência 'f"(x)' = $f'(x)$, ou usar a apóstrofe dentro das aspas duplas "f'(x)". Caso queira escrever aspas duplas, basta usar aspas simples seguidas de duplas: '"sério"' = "sério".

As constantes CHARACTER's são usadas no caso de uma descrição como na tabela 3, usada junto com WRITE, para a informação descritiva do programa.

Uma variável do tipo CHARACTER é uma variável que contem o valor de caracteres.

1.5.4 Digitação de variáveis: Default(Padrão) e Explicit(Explícita)

Quando vemos uma constante, é praticamente fácil identificar se é INTEGER, REAL, ou CHARACTER. Porém, com as variáveis não é tão claro. Como nós e o compilador sabemos o tipo de variável?

Há duas maneiras que o tipo de variável pode ser definida na digitação, a padrão e a explícita. Quando no programa, a variável não está explicitamente especificada, então o padrão é usado:

Default typing

Quaisquer variáveis que comecem com as letras i, j, k, l, m, ou n são automaticamente assumidas como sendo do tipo INTEGER. Quaisquer outras variáveis que comecem com as letras diferentes das supracitadas é assumida como sendo do tipo REAL.

Esse tipo de convenção foi assumida em 1954. O tipo de variável em que se está trabalhando tem que ser explicitamente definida no começo do programa quando se está a fazer as declarações, ex:

INTEGER :: var1[,var2,...,varn]

REAL :: var1[,var2,...,varn]

Onde os valores dentro dos [] são opcionais, neste caso, as variáveis dentro dos colchetes mostram que mais que duas variáveis foram declaradas dentro da mesma linha se eslas estão separadas por vírgulas.

Essas declarações não-executáveis devem sempre vir após o nome do programa e antes dos passos executáveis:

program exemplo
integer :: dia, mes, ano
real :: segundo
...
(Declarações executáveis...)

Como não há *default typing* na variável CHARACTER, pois em 1954 em FORTRAN I, não existia CHARACTER. Todos do tipo CHARACTER devem ser **explícitos!!!** Essa declaração é um pouco mais complicada que as anteriores, mas não é difícil, caso se sinta intimidado a memorizar seus parâmetros é bom sempre ter uma colinha perto, mas com o tempo se torna usual então para aprender não somente FORTRAN mas qualquer outra linguagem o aluno tem de se familiarizar, alguém alguma vez disse que *"a prática leva a perfeição"*, eu digo que a perfeição é inalcançável, mas a prática pode te tornar um profissional!

Como os caracteres tem diferentes tamanhos seu formato é:

character(len=< len >) :: var1[,var2,...,varn]

Onde *len* é o número de caracteres dentro da variável, len=< len > é opcional, sendo que apenas um número aparecer dentro dos parênteses é aceito o tamanho de caracteres, e se não houver parenteses o tamanho do caractere aceito é 1.

character(len=10) :: var
character :: var
character(10) :: var

1.5.5 Mantenha as constantes consistentes nos programa

É importante sempre manter as constantes físicas consistentes durante o seu programa, além de um nome sugestivo, é interessante que o programador não use o valor de $\pi = 3,14$ em uma parte do programa e use $\pi = 3,141593$ em outro momento. Você também deve, escrever as constantes com a máxima precisão aceita pela sua máquina. Se a sua máquina tem 7 dígitos significativos de precisão nos tipos REAL, então deve ser $\pi = 3,141593$ e não $\pi = 3,14$.

Uma boa maneira de melhorar a consistência e precisão é dar um nome fixo para a sua constante e usá-lo durante toda a escrita do código. Um exemplo seria pi = 3,141593.

Nomes de constantes são criadas geralmente usando a atribuição PARAMETER junto a um tipo de declaração:

tipo, parameter :: var1 = valor1[, var2 = valor2,...,varn = valorn]

No caso acima o tipo pode ser INTEGER, REAL, LOGICAL, ou CHARACTER. Mais de uma constante pode ser declarada por linha se for separada por vírgulas.

real, parameter :: pi = 3.141593, e = 2.7182818

Se a constante declarada for do tipo CHARACTER, então não há necessidade de declarar o tamanho da string, já que o programador vai irá a inserir:

character, parameter :: ERROR_MESSAGE = 'Erro desconhecido!'

Dica!
Mantenha a consistência de suas constantes durante seu programa. Para aumentar ainda mais a consistência e entendimento do seu código dê nomes sugestivos a elas afim de chamá-las depois!

Se você entendeu a seção 1.5 hora de testar!!! Caso tenha dificuldades volte e **releia a seção!!!** É extremamente importante que não haja dúvidas!

`As questões de 1-12 contém uma lista de constantes inválidas, se é válida especifique o tipo se for inválida diga o porquê!!!`

Tabela 5: Quiz perguntas! 1- 12

1	10.0
2	-100,000
3	`'that's ok'`
4	123E-5
5	-32768
6	3.14159
7	`"quem é você?"`
8	'3.14159'
9	`'distância =`
10	`"that's ok!"`
11	17.877E+6
12	13.0$\hat{2}$

`De 13-16 contém duas constantes do tipo REAL, diga quando elas são idênticas e o porquê!`

Tabela 6: Quiz perguntas! 13-16

13	4650. ; 465E+3
14	-12.71 ; -1.2E1
15	0.0001 ; 1.0E4
16	3.14159E0 ; 314.159E-3

`A 17-18 tem nomes válido e inválido para o programa, diga qual é qual é qual e o porquê!`

Tabela 7: Quiz perguntas! 17-18

17	`program novo_programa`
18	`program 3rd`

`Entre a 19-23 há uma lista de nomes de variáveis, quais são as válidas e as inválidas?  e o porquê são inválidas?`

Tabela 8: Quiz perguntas! 19-23

19	`length`
20	`distancia`
21	`1problema`
22	`quando_a_aula_termina`
23	`_ok`

`Na 24-25 diga qual é a maneira correta e a não correta de utilizar PARAMETER e o porque.`

Tabela 9: Quiz perguntas! 24-25

24	`real, parameter begin ::  -30`
25	`character, parameter ::  NOME = "Rosa"`

1.6 Declaração de atribuição e cálculos aritméticos

Os cálculos em FORTRAN são especificados com uma **formulação declarativa**, em que a forma mais geral é:

```
nome_var = expressão
```

A declaração de atribuição calcula o valor da expressão que está a direita do sinal de igual, e atribui esse valor a variável a esquerda do sinal de igualdade. É importante entender que o sinal de igual não representa igualde e sim atribuição, ou seja, que é guardado o valor da expressão no local `nome_var`. Por essa razão o sinal de igual (+) é chamado de operador de atribuição. Um exemplo seria:

```
i = i + 1
```

A declaração acima nenhum sentido algébrico, porém em várias linguagens de programação assim como FORTRAN faz total sentido! ela diz para pegar o valor já atribuído e somar mais 1 :).

As expressões a direita do operador de atribuição podem ser quaisquer combinações válidas de constantes, variáveis, parenteses, e operadores aritméticos ou lógicos. Os operadores aritméticos padrões em FORTRAN são:

Tabela 10: Operadores aritméticos básicos em FORTRAN

+	adição
-	subtração
*	multiplicação
/	divisão
**	exponenciação

As seguintes regras aritméticas em FORTRAN devem ser seguidas:

1. Não se deve usar mais de dois operadores um ao lado do outro. Por exemplo, a*-b é errado, deve ser escrito como a*(-b), do mesmo modo é errado escrever a**-2 o correto é a**(-2).
2. Multiplicação implícita, por exemplo, $x(y+z)$ é proibido em FORTRAN. O correto é $x * (y + z)$.
3. Parenteses devem ser usados em agrupamentos de termos quando forem requeridos, exemplo: `2**(8+a)`. A primeira operação a ser realizada será `8+a` e após isso o valor resultante será a potencia de 2.

1.6.1 Aritmética INTEGER

É a aritmética que envolve apenas dados inteiros. Sempre produzirá resultados inteiros, por exemplo, se envolver uma divisão com resultado decimal apenas a parte inteira do número será utilizada:

$$\frac{3}{4} = 0 \qquad \frac{4}{4} = 1 \qquad \frac{5}{4} = 1 \qquad \frac{6}{4} = 1 \qquad \frac{9}{4} = 2$$

Por essa razão INTEGER's nunca devem ser usados para calcular quantidade do mundo real que variam continuamente, tais como, distância, velocidade, tempo, et cetera. Devem ser usados apenas em quantidades naturais como, contadores e índices.

AVISO!
CUIDADO COM O USO DE ARITMÉTICA DE INTEGER'S!!!
ALGUMAS OPERAÇÕES RETORNAM RESULTADOS INESPERADOS

1.6.2 Aritmética REAL

Aritmética de dados REAL's envolve constantes e variáveis reais, sempre produzem resultados reais. Contudo, a aritmética REAL tem suas peculiaridades, como por exemplo, $\frac{1}{3} = 0.\bar{3}$ não pode ser representado completamente pois o computador tem limites. Por essa limitação algumas operações podem não retornar um valor esperado, por exemplo, não pode se afirma o seguinte resultado `3*(1/3) = 1`, mas as vezes a mesma operação pode resultar o esperado, por exemplo, `2*(1/2) = 1`. Testes de igualdade devem ser performados com muita cautela quando estiver trabalhando com números reais.

AVISO!
CUIDADO COM O USO DE ARITMÉTICA DE REAL'S!!!DUAS EXPRESSÕES TEÓRICAS IDÊNTICAS PODERÁ RETORNAR DIFERENTES RESULTADOS

1.6.3 Hierarquia das operações em FORTRAN

Constantemente operações aritméticas são submetidos a apenas uma expressão. Por exemplo a equação de um corpo submetido a uma aceleração:

$$x - x_0 = v_0 + \frac{1}{2}at^2 \mid x_0, v_0 = 0$$

$$x = \frac{1}{2}at^2 \rightarrow \texttt{distancia = 0.5*aceleracao*tempo**2}$$

Encontramos na expressão computacional acima, um operador de atribuição, e na parte de declaração executável, temos duas multiplicações e uma exponenciação. *Citando um meme quando o ex-ministro da justiça e ex-juiz Sérgio Moro quando o mesmo recebeu uma ligação de si mesmo: "Mas como é que pode isso?!"*.

Na expressão acima é importante é importante entender a ordem em que as expressões são realizadas. Se queres que a multiplicação seja feita antes da multiplicação:

```
distancia = 0.5*aceleracao*(tempo**2)
```

E se a exponenciação for feita antes das multiplicações a expressão fica da forma:

```
distancia = (0.5*aceleracao*tempo)**2
```

As duas operações acima resulta em valores diferentes, e devemos ser capazes de distinguir a ambiguidade entre elas. Afim de retirar essa ambiguidade, FORTRAN estabelece uma série de regras onde há a hierarquia que comanda as operações e seus resultados. Geralmente, as operações em fortran seguem as regras de álgebra. Sua hierarquia é:

1. O que estiver dentro dos parênteses sempre será operado primeiro. Começando do parênteses o qual estiver mais interno até o último `(4(3(2(1)2)3)4)`.
2. Todos os exponenciais são feitos da direita para a esquerda. `2**3 = 8; 3**2 = 9.`
3. Todas as multiplicações e divisões são feitas, da esquerda para a direita `4*3/2 = 6; 12/3*4 = 16`.
4. Todas as operações de adição e subtração são feitas, da esquerda para a direita. `3-10+4 = -3; 8+2-12 = -2`.

Avaliando as expressões para distância acima tomando como régua as regras supracitadas, vemos que o tempo dentro do parenteses será feita primeiro e logo após as multiplicações, vindo da esquerda para a direita. Quando todos estão dentro dos parênteses vemos que as multiplicações serão realizadas primeiramente da esquerda para a direita e logo após o resultado será elevado a dois. Algumas pessoas recorrem a certas frases para memorizar, mas a melhor memorização é a prática!!! Mas um exemplo de frase é: Para Expor Meu Doido Aspecto Saboroso, as primeiras letras: P.E.M.D.A.S. = Parênteses, Exponenciação, Multiplicação, Divisão, Adição, Subtração.

escreva um programa e teste isso, brinque com as operações aritméticas, entenda anote, divirta-se e erre!

1.6.4 Aritmética de Modo-Misto

Quando há operações aritméticas entre dois REAL's gera automaticamente um real(Ou flutuante), se é entre dois INTEGER's gera de mesmo modo um inteiro. Geralmente, operações aritméticas são definidas entre números de mesmo tipo. Por exemplo, a adição de de dois números reais é uma operação válida, e de dois inteiros também. Porém, a adição de um inteiro com um real não é uma operação válida, pois, o computador armazena em locais diferentes cada tipo de dado.

O que acontece se fizermos uma operação entre um real e um inteiro? Expressões como essa são chamadas de **operações de modo-misto**, e expressões aritméticas que contém esses dois tipos de dados são chamadas de **aritmética de modo-misto**. Quando há esse tipo de operação o computador converte o INTEGER em um REAL, e é usada a aritmética REAL para fazer o cálculo da expressão.

`Expressão inteira :` $\frac{3}{2} = 1$ `(Resultado inteiro).`

`Expressão real:` $\frac{3.}{2.} = 1.5$ `(Resultado real).`

`Operação de modo-misto:` $\frac{3.}{2} = 1.5$ `(Resultado real).`

Essas regras podem parecer confusas para novos programadores, mas com a prática é certo que esse conhecimento se assentará na memória. E até programadores experientes esquecem dessas regras, porém é importante identificar o erro quando é feito. Tome mais um exemplo:

Tabela 11: Aritmética de modo-misto

Expressão	Resultado
`1 + 1/4`	`1`
`1.+ 1/4`	`1.`
`1 + 1./4`	`1.25`

É interessante ressaltar certos pontos:

- Uma operação entre um INTEGER e um REAL, é chamada de **operação de modo-misto**, uma expressão que contem esse tipo de operação é chamada de **expressão de modo-misto**.
- Quando uma operação de modo-misto é encontrada, FORTRAN converte o inteiro em real e realiza a operação dando um resultado real.
- O modo de conversão automático não acontece até um REAL e um INTEGER aparecer na mesma operação. Portanto, é possível que uma porção de uma expressão seja realizada em INTEGER e outra em REAL.

AVISO!
CUIDADO COM O USO DE ARITMÉTICA DE MODO MISTO SÃO DE DIFÍCIL COMPREENSÃO E PODEM PRODUZIR RESULTADOS ENGANOSOS!!!

Para evitar esses problemas FORTRAN possui 5 tipos de funções que nos permitem controlar explicitamente a conversão entre valores INTEGER's e REAL's. A tabela nos permite entender com funciona essas funções:

Tabela 12: Tabela para funções de conversão

Função	Argumento	Resultado	Comentário
`int(x)`	`real`	`integer`	A parte inteira de x (x é truncado)
`nint(x)`	`real`	`integer`	Inteiro mais próximo de x(x é arredondado)
`ceiling(x)`	`real`	`integer`	Inteiro mais próximo, pra cima ou igual a x
`floor(x)`	`real`	`integer`	Inteiro mais próximo, pra baixo ou igual a x.
`real(i)`	`integer`	`real`	Converte um inteiro em real

1.6.5 Exponenciação na Aritmética de Modo-Misto

Como visto, os modos-mistos são indesejáveis por sua difícil compreensão e por já existir operadores que facilitam esse trabalho. Entretanto, há uma exceção a essa regra: A exponenciação! A exponenciação é desejável para modos-mistos. Para entender o por quê disso, considere o seguinte:

```
resultado = y**n
```

Onde `resultado` e `y` são reais e `n` é um inteiro, o computador encara isso como uma operação real de multiplicar `y` por si mesmo em `n` vezes, ou seja, não é um operação de modo-misto.

Agora, vejamos outro exemplo:

```
resultado = y**x
```

Onde `resultado, y` e `x` são reais, a expressão é desprovida para fazer a operação de multiplicar `y` por ele mesmo `x` vezes. Vamos supor que `x = 2.5`, não há como multiplicar o número por ele mesmo 2,5 vezes... Então uma aproximação algébrica para isso seria:

$$y^x = e^{xlny}$$

Nesse sentido, calculamos $ln(y)$ multiplicamos por x e depois usamos série para aproximar o resultado final. Leva-se um tempo maior para a realização desse cálculo, portanto, sempre que possível eleve a números inteiros.

DICA!
SEMPRE QUE POSSÍVEL USE EXPOENTES INTEIROS.

É mister que se tome nota também que não se deve elevar um número negativo a um outro real, o programa retornará uma mensagem de erro. Pois o logaritmo de um número negativo é indefinido.

AVISO!
NUNCA ELEVE UM NÚMERO NEGATIVO A UMA POTÊNCIA REAL

1.7 Funções Intrínsecas.

Assim como na matemática as funções pegam uma entrada e retornam uma saída, em FORTRAN ocorre a mesma coisa. Quando se trata do meio acadêmico ou mesmo fora dele, precisamos de algumas funções mais complexas que simplesmente de adição, subtração, multiplicação, et cetera. Algumas dessas funções são bem comuns e usadas a todo momento em alguns cursos, outras são mais raras e usadas em problemas mais específicos como também em modelos. Exemplos das funções comuns são, seno, cosseno, tangente, logarítmicas, e raízes quadradas. Exemplos de funções raras, são, funções hiperbólicas, função de Bessel, et cetera.

FORTRAN tem suporte tanto para as funções comuns quanto as raras. Muitas dessas funções foram programadas diretamente na linguagem, e são chamadas de **funções intrínsecas**. As funções menos comum não são inclusas na linguagem, mas o usuário pode chamá-las externamente, são chamadas de **funções externas** ou **funções internas**. As quais serão tratadas mais adiantes nos capítulos posteriores. Uma função em fortran pega um ou mais valores e retorna apenas uma saída. As entradas são conhecidas com **argumentos** e são colocados dentro de parenteses logo após a chamada das funções. A saída é um número, um valor lógico ou uma string , as quais podem ser usadas conjuntamente com outras constantes, variáveis, funções ou expressões. Quando uma função aparece em uma declaração em FORTRAN, os argumentos são levados para um rotina separada onde o valor será computado e o resultado irá substituir a função que fora chamada. Uma lista de funções está a seguir:

Tabela 13: Tabela de listas de algumas das funções intrínsecas do FORTRAN

Função	Valor	Argumento	Resultado	Comentário
`sqrt(x)`	$\sqrt{x}$	`real`	`real`	Raiz quadrada de $x \quad \vert \quad x > 0$
`abs(x)`	$\vert x \vert$	`real/integer`	`real/integer`≥ 0	Retorna o valor absoluto de x
`sin(x)`	$\sin(x)$	`real`	`real`	seno de x
`cos(x)`	$\cos(x)$	`real`	`real`	cosseno de x
`achar(i)`	-	`integer`	`char(1)`	Retorna o caractere na posição `i`
`sind(x)`	$sin(x)$	`real`	`real`	calcula seno em graus
`cosd(x)`	$\cos(x)$	`real`	`real`	calcula o cosseno em radianos
`tan(x)`	$\tan(x)$	`real`	`real`	calcula $tg(x)$
`tand(x)`	$\tan(x)$	`real`	`real`	calcula $tg(x)$ em graus
`exp(x)`	e^x	`real`	`real`	função exponencial
`log(x)`	$\ln(x)$	`real`	`real`	calcula logarítmo natural
`log10(x)`	$\log_{10}(x)$	`real`	`real`	logarítmo na base decimal
`iachar(c)`	-	`char(1)`	`integer`	retorna a posição da letra `c`
`mod(a,b)`	-	`real/integer`	`real/integer`	$a - int\left(\frac{a}{b}\right) \times b$
`max(a,b)`	-	`real/integer`	`real/integer`	Retorna maior valor em `a,b`
`min(a,b)`	-	`real/integer`	`real/integer`	Retorna menor valor em `a,b`
`asin(x)`	$\sin^{-1}(x)$	`real`	`real`	arcoseno de $-1 \leq x \leq 1$
`asind(x)`	$\sin^{-1}(x)$	`real`	`real`	arcoseno de $-1 \leq x \leq 1$ em graus
`acos(x)`	$\cos^{-1}(x)$	`real`	`real`	arcosseno de $-1 \leq x \leq 1$
`acosd(x)`	$\cos^{-1}(x)$	`real`	`real`	arcosseno de $-1 \leq x \leq 1$ em graus
`atan(x)`	$\tan^{-1}(x)$	`real`	`real`	calcula $tg(x) : x \in [-\pi, \pi]$
`atand(x)`	$\tan^{-1}(x)$	`real`	`real`	calcula $tg(x) : x \in [-180^o, 180^o]$
`atan2(x/y)`	$\tan^{-1}\left(\frac{x}{y}\right)$	`real`	`real`	calcula $tg(\left(\frac{x}{y}\right) : x \in [-\pi, \pi]$

1.8 Declarações e Instruções de Entrada e Saida de Fila-Ordenada

Uma **Uma declaração de entrada** lê um ou mais valores e armazena na memória reservada a variável escolhida pelo programador. O dispositivo de entrada, pode ser um teclado interativo ou um arquivo em um ambiente "**batch - lotes**".

Uma **declaração de saída** escreve um valor ou valores, diretamente um um dispositivo de saída, que pode ser a tela do computador ou um arquivo em um ambiente "**batch - lotes**".

Na tabela 3 em `primeiro_programa` nós já vimos declarações de entrada e saída. Onde a declaração de entrada foi: `read(*,*)x,y`. Onde `x,y` são as listas de variáveis ou **Fila-Ordenada de variáveis entrada** do programa, em que os valores lidos serão armazenados. Os parenteses `(*,*)` na declaração contém as informações de controle para a leitura dos dados. O primeiro campo nos parenteses especificam a unidade de entrada/saída, da qual os dados serão lidos(O conceito de entrada/saída será explorado mais adiante, nos capítulos posteriores). O asterisco nesse campo diz que a entrada será lida do dispositivo de entrada padrão do computador - Geralmente o teclado interativo. O segundo campo nos parênteses especifica o formato em que o dado será lido, o asterisco nesse campo significa que será uma entrada em fila-ordenada (também chamado de formato-livre).

Em inglês é comum encontrar o termo como **list-directed input**, que é o mesmo que dizer, quais os tipos de dados determinados que serão os formatos requeridos nos dados de entrada.

Considere o programa abaixo:

```
   program exemplo
integer ::  i,j
real ::  a
character(len=12) ::  chars
read(*,*)i,j,a,chars
end program exemplo
```

Os dados de entrada do programa acima deve conter, dois inteiros, um real e um caractere(string). Logo, os valores devem ser separados um a um por vírgulas ou espaços, ou por linhas. A fila-ordenada da declaração `read` continuará a ler os dados de entrada até os valores preencherem as variáveis uma a uma na fila. Nesse programa podemos aferir as variáveis a seguinte fila:

```
1,2,3.,'esse aqui'
```

Como é lido em fila, a variáveis serão: `i = 1; j = 2; a = 3.; char = 'este aqui'`

Como o tamanho da string é de 12 caracteres e o tamanho da entrada é apenas 9, os outros 3 são espaços adicionados automaticamente, é bom tomar nota que se tiver espaço entre as palavras é necessário que a string esteja dentro de aspas. Quando estiver montando o programa usando fila-ordenada, **ORDENE!!!** Ou seja, coloque os dados refentes aos tipos de variáveis em que estiver trabalhando.

Na leitura ordenada, cada declaração `read` irá ler os dados a partir de uma nova linha e qualquer outro valor que estiver nessa linha será ignorado, tome o próximo exemplo:

```
   PROGRAM exemplo_2
INTEGER ::  i, j, k, l
READ (*,*) i, j
READ (*,*) k, l
END PROGRAM exemplo_2
```

Considere a entrada:

```
1, 2, 3, 4

5, 6, 7, 8
```

O primeiro `read` irá ler 1 e 2, e há de ignorar 3 e 4. O segundo `read` irá ler 5 e 6, e há de ignorar 7 e 8.

Uma boa ideia é *ecoar* a variável, ou seja, fazer o programa escrevê-la na tela assim que for digitada usando o `write`, para saber se o número digitado foi o correto e entender se houver algum erro, esse erro pode ser do usuário.

DICA!

ECOE SUAS VARIÁVEIS DURANTE A EXECUÇÃO DO PROGRMA!!!

`write(*,*) saida_fila_var`

INFORMAÇÃO!

HÁ UMA OUTRA MANEIRA DE ESCREVER NA TELA SUAS VARIÁVEIS, COMANDO PRINT*,

`print*, saida_fila_var`

O termo **saída em Fila-Ordena**, significa que os tipos de valores na lista de saída da declaração `write` determina o formato dos dados de saída. Considere o programa abaixo:

```
   PROGRAM saida_exemplo
INTEGER ::  ix
REAL ::  theta
ix = 1
test = .TRUE.
theta = 3.141593
WRITE (*,*) 'IX =', ix
WRITE (*,*) 'THETA =', theta
WRITE (*,*) ' COS(THETA) =', COS(theta)
WRITE (*,*) REAL(ix), NINT(theta)
END PROGRAM saida_exemplo
```

A saída será:

```
   IX = 1
```

```
THETA = 3.141593

COS(THETA) = -1.000000

1.000000    3
```

Esse exemplo ilustra inúmeras características sobre a declaração em fila-ordenada:

1. A lista de saída pode conter constantes (`'IX = '` é uma constante), variáveis, funções e expressões. Em todos os casos, esses dados são impressos no dispositivo de saída padrão.
2. A formatação dos dados de saída casa com o tipo de valor que está sendo impresso. Por exemplo, mesmo `theta` sendo uma variável do tipo `real`, `nint(theta)` é do tipo `integer`. Por isso, a sexta declaração `write` produz uma saída do número 3 (O mais próximo inteiro de 3,141593)
3. A saída de fila-ordenada feita pela declaração `write` não é bonita! Haja vista que os valores impressos não se organizam em colunas ordenadas, como também não há como de algarismos significativos que serão impressos para os números reais. Porém, posteriormente falaremos sobre como formatar a saída.

QUESTIONÁRIO 2-3:

Esse questionário é um teste para avaliar os conhecimentos adquiridos e averiguar se os conceitos foram aprendidos durante a seção **2.7** e **2.8**, se houver problemas ao fazer o questionário volte e releia as seções! É imprescindível que as bases fiquem bem estabelecidas para você, o conhecimento é um desenvolvimento processual como também pessoal!!!

Converta as expressões algébricas para para FORTRAN:

1. A resistência equivalente quando conectada em série, de quatro resistores $\{R_1, R_2, R_3, R_4\}$, R_{eq}:

$$R_{eq} = R_1 + R_2 + R_3 + R_4$$

2. A resistência equivalente R_{eq} de quatro resistores $\{R_1, R_2, R_3, R_4\}$ conectados em paralelos:

$$R_{eq} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}}$$

3. O período T de um pêndulo, onde L é o comprimento do fio e g é a constante da aceleração gravitacional.:

$$T = 2\pi\sqrt{\frac{L}{g}}$$

A equação de um oscilador senoidal amortecido, onde V_m é o valor máximo da oscilação, α é o fator expoente de amortecimento e ω é a velocidade angular da oscilação:

$$v(t) = V_M e^{-\alpha t} \cos(\omega t)$$

Converta as seguintes operações declarativas de FORTRAN para equações algébricas:

5. O movimento de um objeto em um campo gravitacional constante:

```
distancia = 0.5*aceler*t**2 + vel_0*t + pos_0
```

6. A frequência de oscilação em um circuito RLC amortecido:

```
freq = 1./(2.*PI*sqrt(1*c))
```

7. Energia armazenada em um indutor:

```
energia = 1.0/2.0*indutancia*corrente**2
```

8. Quais valores serão impressos no seguinte programa?

```
   PROGRAM questionario_1
INTEGER ::  i
REAL ::  a
a = 0.05
i = NINT(2.  * 3.141493 / a)
a = a * (5 / 3)
WRITE (*,*) i, a
END PROGRAM questionario_1
```

9. Com a entrada dada abaixo, o que será impresso no seguinte programa?

```
   PROGRAM quiz_2
INTEGER ::  i, j, k
REAL ::  a, b, c
READ (*,*) i, j, a
READ (*,*) b, k
c = SIN ((3.141593 / 180) * a)
WRITE (*,*) i, j, k, a, b, c
END PROGRAM questionário_2
```

Entrada:

```
   1, 3
2., 45., 17.
30., 180, 6.
```

1.9 Inicialização de Variáveis

Considere o programa a seguir:

```
   PROGRAM init
INTEGER ::  i
WRITE (*,*) i
END PROGRAM init
```

Qual o valor guardado na variável `i`? O que será impresso na declaração `write`. A resposta é **NÃO SABEMOS!!!** A variável `i` é um exemplo de **variável não iniciada**. Foi definida por `integer ::  i`, mas nenhum valor foi atribuído a variável. O valor de qualquer variável indefinida não está definida no padronização da linguagem. Alguns compiladores simplesmente atribuem a variável o valor zero, e alguns deles escolhem diferentes padrões. Compiladores antigos atribuíam valores da última variável! Outros compiladores apresentam uma mensagem de **erro**.

Variáveis não inicializadas podem ser/apresentar-se como um enorme problema tendo em vista que são apresentadas de diferentes formas em diferentes máquinas, ou seja, o programa funciona bem em um computador e falha quando rodado em outro. Até em uma mesma máquina, o programa pode funcionar algumas vezes e falhar em outras, pois depende dos dados deixados na mesma memória ocupada por um programa anteriormente. Tal gambiarra é totalmente inaceitável(*não que gambiarras sejam ruins, particularmente eu as adoro como também sou profissional na área tendo em vista que elas são um demonstrativo direto da criatividade do brasileiro S2. justamente por eu conhecê-las essa é inaceitável*).

DICA!
SEMPRE INICIALIZE SUAS VARIÁVEIS NO INÍCIO DO PROGRAMA!!!

Existem 3 técnicas disponíveis para inicializar as variáveis em FORTRAN:

Uma declaração de atribuição, declaração `read`, e inicialização do tipo de instrução declaração. *Uma quarta técnica é antiga e consistia em usar declarações de dados. Essas declarações foram deixadas para trás tem compatibilidade com versões antigas de FORTRAN, mas que tem sido substituída pela inicialização do tipo instrução de declaração. Declaração de dados não deve ser usadas para a construção de novos programas, esse tipo de inicialização será vista nos capítulos posteriores.*

Uma declaração de atribuição, atribui o valor da expressão a variável do lado direito do sinal de igualdade, um exemplo a seguir:

```
   PROGRAM init_1
INTEGER ::  i
i = 1
WRITE (*,*) i
END PROGRAM init_1
```

A declaração `read`, deve ser usada para inicializar as variáveis pelo usuário do código que dará uma entrada a variável. Diferentemente da declaração de atribuição o usuário pode modificar o valor da variável durante a execução do programa. Um exemplo a seguir:

```
   PROGRAM init_2
INTEGER ::  i
READ (*,*) i
WRITE (*,*) i
END PROGRAM init_2
```

A terceira técnica disponível para inicializar as variáveis em FORTRAN é definir o valor inicial no tipo na instrução de declaração do código. Esta instrução especifica que um valor deve ser pré carregado dentro de uma variável durante a compilação e fazendo o link no processo. Note a diferença fundamental entre a declaração de atribuição e a instrução de declaração.

A instrução de declaração inicia a variável antes do programa começar a rodar, enquanto a declaração de atribuição inicia a variável durante a execução.

A forma da instrução de declaração é:

```
type ::  var1 = valor, [var2 = valor 2 ,...,]
```

Qualquer número de variáveis pode ser declarado e inicializado em um único tipo de instrução de declaração que esteja entre vírgulas. Um exemplo de instrução de declaração usado para iniciar as variáveis:

```
   REAL ::  time = 0.0, distance = 5128.
INTEGER ::  loop = 10
```

1.10 A Declaração Implicit None

Ainda há, uma declaração não-executável extremamente importante! A declaração **IMPLICIT NONE**. Quando usada, desarma todas as provisões padrão do FORTRAN. Quando o `implicit none` é incluído em um programa, qualquer variável que não é declarada explicitamente é considerada como um erro. Essa declaração tem de vir antes de qualquer outra na próxima linha após a inicialização do programa.

Quando usada o programador é obrigado a explicitar suas variáveis. Num primeiro pensamento isso parece ser desvantajoso, já que o programador tem mais trabalho quando escreve o programa. Essa impressão não poderia estar mais errada!!! De fato, há inúmeras vantagens em usar essa declaração.

A maioria dos erros nos programas são erros de tipografias, e a declaração `implicit none` captura esses erros durante a compilação antes que possam surgir mais erros a partir destes anteriores. Um exemplo, o próximo programa:

```
   PROGRAM test_1
REAL ::  time = 10.0
WRITE (*,*) 'Time = ', tmie
END PROGRAM test_1
```

No programa acima a variável `time` está escrita errada `tmie`, quando o programa for compilado irá aparecer na tela:

```
Time = 0.000000E+00
```

Quando a declaração `implicit none` for colocada no programa, ele emitirá uma mensagem de erro que varia de máquina para máquina:

```
   1 PROGRAM test_1
2 IMPLICIT NONE
3 REAL ::  time = 10.0
4 WRITE (*,*) 'Time = ', tmie
.......................1
(1) Error:  This name does not have a type, and must have an explicit type.
[TMIE]
5 END PROGRAM
```

Ao invés de ter uma uma resposta errada em um programa que funcionava, agora temos uma mensagem de erro explicita expondo o problema durante a compilação. **ISSO É UMA ENORME VANTAGEM QUANDO TRABALHAMOS COM PROGRAMAS LONGOS QUE CONTÉM MUITAS VARIÁVEIS!!!** Uma outra vantagem do `implicit none` é que ele torna o programa mais instável. Qualquer programa que usa esse tipo de declaração, deve ter uma lista de variáveis contendo todas as variáveis usadas. Se o programa precisar de ser modificado, então o programador pode checar a lista de variáveis para evitar usar variáveis de mesmo nome durante o programa. Essa checagem ajuda a eliminar um erro muito comum, na qual a modificação do programa admite uma troca nos valores de algumas variáveis usadas em algumas partes do programa.

Em geral, o uso do `implicit none` se torna cada vez mais vantajoso ao longo que o projeto de programação aumenta significativamente.

DICA!

SEMPRE EXPLICITE E DEFINA CADA VARIÁVEL NO SEU PROGRAMA. USE SEMPRE O `IMPLICIT NONE` PARA AJUDAR A PONTUAR E CORRIGIR OS ERROS DE TIPOGRAFIA ANTES QUE ELE SE TORNEM ERROS DE EXECUÇÃO!!!

1.11 EXEMPLOS DE PROGRAMAS

Nesta subseção iremos apresentar uma série de exemplos em FORTRAN, conceitos para escrever programas funcionais e simples. Apresentaremos alguns exemplos em que esses conceitos são utilizados.

1.11.1 Conversão de Temperatura:

faça um programa que lê a temperatura em graus Fahrenheit e converte na temperatura absoluta Kelvin.

$$T_{Kelvin} = \left(\frac{5}{9} T_{(^oF)} - 32.0\right) + 273.15 \tag{1}$$

1. O usuário terá que entrar com a temperatura em Fahrenheit
2. Ler a temperatura de entrada
3. Calcular a temperatura a partir da equação 1
4. Escrever um resultado e parar.

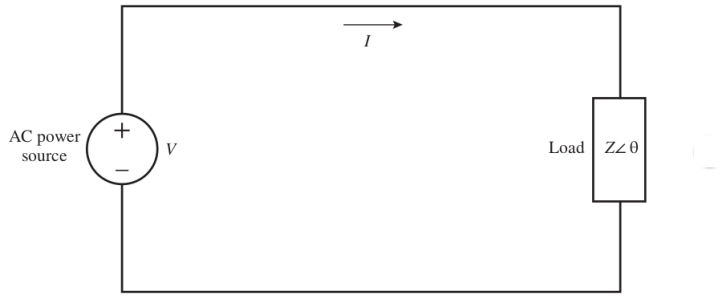
```
   PROGRAM temp_conversion
!  Purpose:
!To convert an input temperature from degrees Fahrenheit to
!an output temperature in kelvins.
!
!  Record of revisions:
!Date Programmer Description of change
!==== ========== ====================
!11/03/15 -- S. J. ChapmanOriginal code
!
IMPLICIT NONE ! Force explicit declaration of variables
!  Data dictionary:  declare variable types, definitions, & units
REAL ::  temp_f !  Temperature in degrees Fahrenheit
REAL ::  temp_k !  Temperature in kelvins
!  Prompt the user for the input temperature.
WRITE (*,*) 'Enter the temperature in degrees Fahrenheit:  '
READ (*,*) temp_f
!  Convert to kelvins.  temp_k = (5.  / 9.)  * (temp_f - 32.)  + 273.15
!  Write out the result.
WRITE (*,*) temp_f, ' degrees Fahrenheit = ', temp_k, ' kelvins'
!  Finish up.
END PROGRAM temp_conversion
```

No Programa acima foram escolhidas adequadamente as unidades físicas. Sempre faça isso!!!

1.11.2 Cálculo da Real, Reativa e Potência Aparente

A figura 2 mostra uma fonte AC senoidal com voltagem V suprindo uma impedância de carga $Z < \theta\Omega$. De uma simples teoria de circuitos, a corrente r.m.s. I, a potência real P, a potência reativa Q, a aparência S e o fator potência PF fornecida pela carga dada pelas equações a seguir:

Figura 2: Fonte senoidal AC com voltagem V suprindo uma impedância de carga $Z < \theta\Omega$



$$V = IR$$
$$P = VI\cos\theta$$
$$Q = VI\cos\theta$$
$$S = VI$$
$$PF = \cos\theta$$

Onde V é a voltagem r.m.s. da fonte de potência em volts(V). A unidade de corrente é ampere(A), da potência real watts(W), da potência reativa volts-amperes-reativos(VAR), e da potência aparente volt-ampere(VA). O fator potência não tem unidades associados a ele.

Dada a voltagem em r.m.s. da potência da fonte, da magnitude e do ângulo da impedância Z, escreva um programa que calcule a corrente r.m.s. I, a potência real P, um poder reativo Q, potência aparente S, e o fator potência PF da carga.

RESOLUÇÃO:

Nesse programa necessitamos de ler a voltagem r.m.s. V da fonte, e da magnitude Z e ângulo θ da impedância. A entrada da fonte de voltagem será mensurada em volts V, a magnitude da impedância Z em *ohms*, e o ângulo θ é em graus. Uma vez que os dados são lidos, devemos converter o ângulo de graus para radianos para o uso em funções trigonométricas do FORTRAN. Após, os valores desejados devem ser calculados, e os resultados devem ser impressos.

O programa deve performar nos seguintes passos:

1. Pedir ao usuário os valores para entrar com a voltagem em volts
2. Ler a fonte de voltagem
3. Pedir ao usuário para entrar com a magnitude e ângulo de impedância em *ohms* e graus.

4. Ler a magnitude do ângulo de impedância
5. Calcular a corrente I da primeira equação
6. Calcular a potência real através da segunda equação.
7. Calcular a potência reativa da terceira da terceira equação
8. Calcular a potência aparente S da quarta equação
9. Calcular o fator potência PF na quinta equação
10. Escrever o resultado e parar o programa.

```
   PROGRAM power
!
!  Purpose:
!Calcular a corrente, real, reativa e potência aparente
!como também o fator de potência usado para uma carga
!
!  Dados do programa:
!Data Programador Descricao mudanca
!==== =========== ====================
!11/03/15 S. J. Chapman Original code
!
IMPLICIT NONE

   !  Dicionario de dados:  declarar constantes
REAL,PARAMETER ::  DEG_2_RAD = 0.01745329 !  Deg to radians factor
!  Dicionario de dados:  declarar variaveis, definicoes, & unidades
REAL ::  amps !  corrente na carga (A)
REAL ::  p !  potencia real da carga (W)
REAL ::  pf !  fator potencia da carga (admensional)
REAL ::  q !  potencia reativa da carga (VAR)
REAL ::  s !  potencia aparente da carga (VA)
REAL ::  theta !  Angulo de impedancia da carga (deg)
REAL ::  volts !  voltagem rms da fonte de potencia (V)
REAL ::  z !  magnitude da impedancia de carga(ohms)

   !  pedir ao usuario a voltagem em rms.
WRITE (*,*) 'Entre com a voltagem rms da fonte:  '
READ (*,*) volts
!  Pedir ao usuario a magnitude e angulo da impedancia.
WRITE (*,*) 'Entre com a magnitude e angulo de impedancia '
WRITE (*,*) 'em ohms e em graus:  '
READ (*,*) z, theta
!  Fazer os calculos
amps = volts / z !Rms current
p = volts * amps * cos (theta * DEG_2_RAD) !Potencia real
q = volts * amps * sin (theta * DEG_2_RAD) !Potencia reativa
s = volts * amps !potencia aparente
pf = cos ( theta * DEG_2_RAD) !fator potencia
```

```
   !  Saida dos resultados.
WRITE (*,*) 'Voltagem =', volts, ' volts'
WRITE (*,*) 'Impedancia =', z, ' ohms at ', theta,' graus'
WRITE (*,*) 'Corrente =', amps, ' amps'
WRITE (*,*) 'Potencia real =', p, ' watts'
WRITE (*,*) 'Potencia reativa =', q, ' VAR'
WRITE (*,*) 'Potencia aparente =', s, ' VA'
WRITE (*,*) 'Fator potencia  =', pf

   !  Finalizar
END PROGRAM power
```

1.12 Desbugar os Programas em FORTRAN

Há um ditado antigo que diz que a única certeza que temos na vida é a morte e os impostos...LOOL... Podemos adicionar uma terceira, se você escrever um programa de qualquer tamanho significativo, **ELE NÃO IRÁ RODAR DE PRIMEIRA...(TROLADO!!!)**. Erros em programas são conhecidos como **bugs** e o processo de encontrar e eliminar esses erros é conhecido como **desbugar**(*já dizia o professor procópio "problema bugante!!!"*). Se estamos escrevendo um programa e ele não funciona, como desbugá-lo-emos?(essa mesóclise casada com neologismo foi digna do ex-presidente da república Michel Temer #vampiraoqueeusei).

Três tipos de erros muito comuns são:

- **Erro de sintaxe!!!**

  São erros nas declarações em si, como trocar nome de variável ou pontuações. Esses são detectados pelo compilados durante a compilação.

- **Erros no tempo de execução!!!**

  Ocorre quando uma há uma operação matemática ilegal, por exemplo, dividir por zero. Esses erros fazem o programa abortar sua execução na hora.

- **Erros lógicos!!!**

  Erros lógicos é quando o programa compila e funciona, porém, dá respostas erradas!!!

Os erros mais comuns são os de tipografia, alguns deles criam declarações inválidas em FORTRAN, saímos da maioria deles usando `implicit none`. De qualquer forma, se, uma variável válida for substituída por outra variável válida de nome diferente, o compilador não pode detectar esse erro, esse tipo de substituição pode ocorrer quando há duas variáveis de nome parecido.

Se duas variáveis `vel1` e `vel2` são usadas, é provável em algum momento confundir as duas e uma ser tomada indevidamente como a outra. Esse tipo de erro produzirá erros lógicos e você terá de checar o código inteiro até achá-lo.

Algumas vezes mesmo com erros no tempo de execução o programa roda, isso quer dizer que existe problema não somente no tempo de execução, mas também há ou no dado de entrada ou na estrutura lógica do programa. O primeiro passo em localizar esses bugs é checar os dados de entrada do programa. Seu programa deve ter sido feito para sempre ecoar esses dados, se não, volte nele e escreva todos os `write` para ele ecoá-las e você encontrá-las. Se o nome das variáveis estiverem corretas após a checagem, então provável que seja um erro lógico, você deve checar todas as suas atribuições:

- Se as declarações de atribuição forem muitas, divida-as em blocos menores, pois assim será mais fácil a verificação.

- Faça a checagem de todos os parênteses se eles se encontram nos locais corretos.

- Tenha a certeza que todas as suas variáveis foram inicializadas corretamente!!!

- Esteja certo de que todas as funções estão nas unidades corretas, um exemplo, a entrada das funções trigonométricas são em radianos não em graus.

- Se você fez o uso da aritmética de modo-misto, confira se há possíveis erros associados.

Se mesmo assim o erro continuar, adicione inúmeras declarações `write` ao longo do programa e veja o resultado dos cálculos intermediários, se achar o ponto de erro de cálculo, então estás olhando para um erro do programa, o que é 95% da batalha ganha!!!

Se ainda não consegues achar o problema, vai dar uma volta, jogar conversa fora, fazer algo que distraia a cabeça, sei lá uma trepada e depois volte pois as vezes o erro pode estar na nossa cara, mas como nosso sistema biológico quando pressão responde ao movimento é provável que você não ache com a cabeça cheia... Ou então mande o código a um colega, pois achar o erro alheio é muito mais fácil que achar o próprio, estatisticamente falando...

DICA!
*** USE A DECLARAÇÃO IMPLICIT NONE**
*** ECOE TODAS AS VARIÁVEIS**
*** INICIALIZE TODAS AS VARIÁVEIS**
*** USE OS PERENTESES DE FORMA CORRETA**

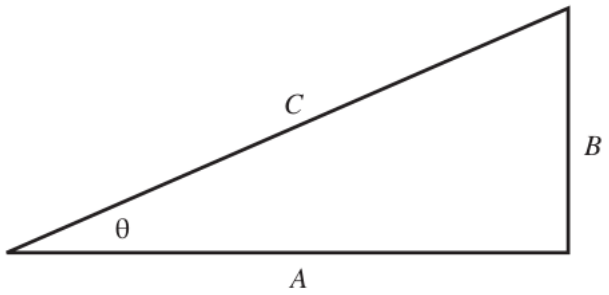
Todos os compiladores modernos possuem ferramentas desbugantes que se chamam *symbolic debuggers*, que te permite andar pelo programa uma declaração por vez e avaliar o valor das variáveis de cada passo ao longo do caminho. *Symbolic debuggers* te permite ver todos os resultados intermediários sem a necessidade de colocar `write` no seu código. São poderosos e flexíveis, mas não são padronizados, ou seja, cada compilador tem um padrão diferente do outro.

1.13 Exercícios:

Nos exercícios a seguir, alguns dou a fórmula outros não. Pois, quando estiver diante de uma situação real ninguém te dará formula alguma, portanto, tens que tirá-las do bolso! Entretanto, este é o primeiro capítulo e será o único que ainda fornecerei algumas fórmulas. Ache a expressão algébrica e a traduza para FORTRAN. Se proponha problemas também!!!

1.13.1 A figura 3 apresenta um triângulo retângulo, usando as relações trigonométricas básicas, escreva 7 programas. Os primeiros para avaliar o valor de A,B e C, dado 1 dos lados e o angulo θ. Outros para calcular os lados dado dois lados usando a relação de Pitágoras. O sétimo programa escreva uma livre para calcular a área do triângulo tendo 1 dos lados que não seja B e o ângulo θ

Figura 3: Triângulo Retângulo



1.13.2 Usando as fórmulas abaixo, e considerando a conservação da energia, calcule a energia mecânica total de um objeto de massa m no campo gravitacional da terra, considere $g = 9.8ms^{-2}$, a massa m e a altura h será dada pelo usuário. Escreva outro programa que calcule a velocidade final desse objeto em queda livre e um que determine o tempo de queda.

$$E_K = \frac{1}{2}mv^2$$

$$E_U = mgh$$

$$E_{mec} = E_K + E_U$$

1.13.3 Calcule o período de um pêndulo usando a formula abaixo e usando trigonometria calcule também a posição que ele se encontra no durante esse período. A formula abaixo será dada apenas para o período. L é dado pelo usuário e considere $g = 9.8ms^{-2}$

$$T = 2\pi\sqrt{\frac{L}{g}}$$

Dica: Faça o desenho do pêndulo, faça as relações trigonométricas para encontrar o tamanho do setor e onde o pêndulo se encontra.

1.13.4 Um corpo se em regime de movimento circular uniforme, ou seja, uma velocidade tangencial constante, a sua aceleração radial é $\alpha = \frac{v^2}{r}$ onde v é a velocidade tangencial e r o raio, lembre se das unidades!!! Suponha que o corpo seja uma aeronave faça um programa que dê as seguintes informações:

***Suponha que a velocidade é de $Mach 0.80$, seja, 80% da velocidade do som, se a aceleração centrípeta é $2.5g$ qual o tamanho do raio?**
Use $\{1Mach = 340m/s, 1g = 9.81m/s\}$

***Agora que a velocidade seja de $Mach 1.5$, qual será o raio?**

***Se a aceleração máxima que o piloto consegue suportar é $7g$, qual o menor raio possível a uma velocidade de $Mach 1.5$?**

1.13.5 A velocidade de escape de um corpo com uma massa M e raio R é dado pela fórmula abaixo, considerando a tabela abaixo com a massa e raio de alguns astros e retorne a velocidade de escape desses corpos:

Tabela 14: Massa e raio de alguns astros

Planeta	Massa(kg)	Raio(m)
Terra	6.0×10^{24}	6.4×10^{6}
Lua	7.4×10^{22}	1.7×10^{6}
Ceres	8.7×10^{20}	4.7×10^{5}
Jupiter	1.9×10^{27}	7.1×10^{7}

Velocidade de escape

$$v_e = \frac{\sqrt{2GM}}{R}$$

**Use $G = 6.673 \times 10^{-11} Nm^{-2}kg^{-2}$.

1.14 REFERÊNCIAS:

1. Fortran for scientists and engeneers - Chapman, S.
2. Getting started with Fortran - Anne Fouilloux - GitHub
3. Introduction to Programming With Fortran 95/2003/2008 - Ed Jorgesen